A Matching Algorithm for Measuring the Structural Similarity between an XML Document and a DTD and its Applications*

Elisa Bertino[a], Giovanna Guerrini[b] and Marco Mesiti[a]

[a]Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano. Via Comelico 39/41, 20135 Milano - Italy. {bertino,mesiti}@dico.unimi.it

[b]Dipartimento di Informatica, Università degli Studi di Pisa. Via Buonarroti 2, 56127 Pisa - Italy. guerrini@disi.unige.it

In this paper we propose a matching algorithm for measuring the structural similarity between an XML document and a DTD. The matching algorithm, by comparing the document structure against the one the DTD requires, is able to identify commonalities and differences. Differences can be due to the presence of extra elements with respect to those the DTD requires and to the absence of required elements. The evaluation of commonalities and differences gives raise to a numerical rank of the structural similarity. Moreover, in the paper, some applications of the matching algorithm are discussed. Specifically, the matching algorithm is exploited for the classification of XML documents against a set of DTDs, the evolution of the DTD structure, the evaluation of structural queries, the selective dissemination of XML documents, and the protection of XML document contents.

Key words: Structural similarity, XML document handling and querying, document classification, structure evolution, structural queries, selective dissemination of documents, document protection.

1. Introduction

Similarity plays a crucial role in many research fields. Similarity serves as an organization principle by which individuals classify objects, form concepts, and make generalization [30]. Similarity can be computed at different layers of abstraction: at data layer (i.e. similarity between data), at type layer (i.e. similarity between types – also referred to as schema, models, or structures, depending on the application domain) or between the two layers (i.e. similarity between data and types). Evaluating similarity among data is relevant for creating clusters of information related to the same topic and for ranking them. For example, in the image field, the similarity measure can be exploited for grouping together images containing the same subject. Evaluating similarity between types is relevant for the integration of schema describing the same kind of information but using different structures [2] and for schema clustering [19]. Evaluating similarity between data and types is relevant for identifying a data generator, and thus, applying to data the properties specified for the type. Orthogonally to this classification, similarity can be focused on contents or on the structures of data involved.

In the XML [32] arena, the possibility of evaluating similarity has been receiving a lot of attention because more and more information exchanged on the Web is adhering to this format and applications need to retrieve, access, and handle XML documents imposing relaxed conditions and returning approximate results. At the data layer, many approaches have been developed for measuring the similarity among XML documents in order to cluster together documents dealing with the same topic. Standard approaches consider the textual content of the documents [1], whereas, recently, some new approaches consider also the structure of docu-

*A preliminary version of this paper appeared in Proc. of 13^{th} *International Syposium on Methodologies for Intelligent Systems*, 2002, with the title "Matching an XML Document against a Set of DTDs".

ments [12,22]. For what concerns structural similarity, many approaches rely on the hierarchical structures of documents exploiting evaluation functions based on the tree edit distance [25]. At the type layer, other approaches have been developed for the integration of schemas that represent the same kind of data [9,10,20] and for schema clustering [19].

Despite this huge activity at data and type layers and the attractive potential applications in many fields, no efforts have been devoted to the computation of structural similarity between an XML document (the data) and a schema (the type). In this paper we introduce a matching algorithm for computing the structural similarity between an XML document and a DTD, which is the simplest means by which structural properties of an XML document can be specified.

In matching a document against a DTD, some attributes and subelements specified for an element in the DTD can be missing from the corresponding element of the document, and, vice versa, the document can contain some additional attributes and subelements not appearing in the DTD. Moreover, since we are focusing on data-centric documents, elements/attributes in the document can follow a different order w.r.t. the one specified in the DTD. Finally, document and DTD tags may not be exactly the same, provided they are *stems* or are similar enough according to a given Thesaurus. Therefore, tag similarity rather than tag equality is supported. In matching a document against a DTD the goal is then to quantify, through an appropriate measure, the structural similarity between the document and the DTD. Though our technique handles all features of XML documents, in the paper we focus on the most meaningful *core* of the approach, thus we restrict ourselves to a subset of XML documents and to tag equality. We refer the interested reader to [6,21] for the general case.

Many applications can be devised for the matching algorithm. For example, in the exchange of XML documents on the Web it is not always possible to force a database to adhere or to integrate its schema with other schemas describing the same kind of data. Therefore, the matching algorithm can be employed for computing the similarity between documents arriving at a given XML database and the local schema. As another example, the possibility to exploit the structure of documents for their retrieval is pushing the need for query engines able to evaluate structural queries (i.e. queries in which conditions are imposed on the structure of the required documents). The query engines can employ the matching algorithm for evaluating the similarity between a document (possible answer of the query) and a structural query represented as a schema plus content conditions. By means of this, the query engine can filter and rank answers to the query.

In this paper we focus on five applications of the algorithm: (1) the classification of XML documents against a set of DTDs; (2) the generation of a new schema for a DTD by extracting structural information during the classification of XML documents; (3) the development of an XML-based search engine able to answer approximate structural queries; (4) the selective dissemination of XML documents; (5) the protection of the contents of documents classified against a set of DTDs of a database, by propagating the authorization policies specified at DTD level.

The remainder of the paper is organized as follows. Section 2 presents our tree representation for XML documents and DTDs. Section 3 discusses the basic principles underlying the behavior of the matching algorithm. Section 4 discusses in details the matching algorithm, whereas Section 5 presents the matching algorithm applications. Section 6 discusses related work, and, finally, Section 7 concludes the work and outlines future research directions.

2. Documents and DTDs as Trees

A key feature of XML is represented by the various options one has available when modeling document subelements. We illustrate those options by means of the document and DTD reported respectively in Figures 1 and 2. The DTD in the figure shows that for each subelement it is possible to specify whether it is optional ('?'), whether it may occur several times ('*' for 0 or more times, and '+' for 1 or more times), whether some subele-

```
<product>
  <name>Deliver</name>
  <urls>
   <download> http://.../deliver.tgz </download>
   <homepage> http://.../index.html </homepage>
  </urls>
  <description>Mail... </description>
  <author>
   <fName>Chip</fName>
   <lName>Salzenberg</lName>
  </author>
  <version>2.1.13</version>
</product>
```

Figure 1: An example of XML document

```
<!DOCTYPE product[
<!ELEMENT product(name,urls+,description,
          (author*|vendor),version?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT urls(download*, homepage)>
<!ELEMENT description ANY>
<!ELEMENT author (fName, mName?, lName)>
<!ELEMENT vendor (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT download (#PCDATA)>
<!ELEMENT homepage (#PCDATA)>
<!ELEMENT fName (#PCDATA)>
<!ELEMENT mName (#PCDATA)>
<!ELEMENT lName (#PCDATA)> ] >
```

Figure 2: An example of DTD

ments are alternative with respect to each other ('|') or are grouped in a sequence (','). We focus on a subset of XML documents. Specifically, we only consider elements (that can have a nested structure) disregarding attributes (that can be seen as a particular case of elements). Since we disregard attributes, we only consider nonempty elements. However, empty elements can be simply handled as `#PCDATA` elements with the constraint to have a null content.

In the matching process, we represent both DTDs and XML documents through labeled trees. The document representation is compliant with the tree representation of DOM [31]. By contrast, the DTD representation makes easy the description of the algorithms.

2.1. Tree Representation of Documents

An XML document is represented as a labeled tree. This representation only relies on information determined from the structure of the document. Our definition is based on the classical definition of labeled tree. We recall that, given a set $\mathcal{N}$ of nodes, a tree is defined by induction as follows: $v \in \mathcal{N}$ is a tree; if $T_1, \ldots, T_n$ are trees, then $(v, [T_1, \ldots, T_n])$ is a tree.[2] Let $N(T) \subseteq \mathcal{N}$ denote the set of nodes of a tree T, and given a set $\mathcal{A}$ of labels, a labeled tree is a pair (T, φ), where T is a tree, and φ is a labeling function s.t. $\forall v \in N(T),\ \varphi(v) \in \mathcal{A}$. In our representation of documents each node represents an element tag or a value. The labels used to label the tree belong to a set of element tags ($\mathcal{EN}$) and to a set of values that the data contents of an element can assume ($\mathcal{V}$). In each tree representing a document the label of the root belongs to $\mathcal{EN}$ (it is the name of the document element). Moreover, leaves of the tree are labeled by values in $\mathcal{V}$.

Definition 1 (*XML document*). An XML document is a labeled tree (D, φ_D) defined on the set of labels $\mathcal{EN} \cup \mathcal{V}$ with the following properties:

1. $D = (v, C)$ with $\varphi_D(v) \in \mathcal{EN}$;
2. for each subtree (v, C) of D, $\varphi_D(v) \in \mathcal{EN}$;
3. for each (leaf) subtree v of D, $\varphi_D(v) \in \mathcal{V}$.□

For the sake of simplicity, in the graphical representation we omit the explicit direction of edges. All edges are oriented downward. Figure 3 shows the tree representation of the XML document in Figure 1.

2.2. Tree Representation of DTDs

A DTD is also represented as a labeled tree. In the tree representation, in order to represent optional elements, repeatable elements, sequence and alternative of elements, the set of operators $\mathcal{OP} = \{$`?`, `*`, `+`, `AND`, `OR`$\}$ is introduced. The `AND` operator represents a sequence of elements, the `OR` operator represents an alternative of elements (exactly one of the alternatives must be selected),

[2] In the remainder of the paper, for sake of simplicity, we denote with C the subtrees of T (i.e. $C = [T_1, \ldots, T_n]$) when it is only relevant to know that T is an internal subtree of a DTD.

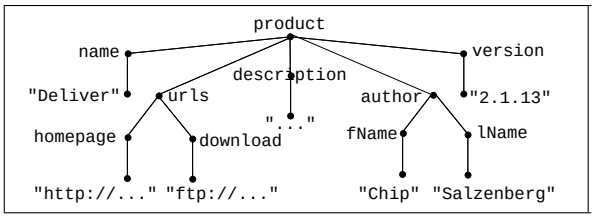

Figure 3: Tree representation of the XML document in Figure 1

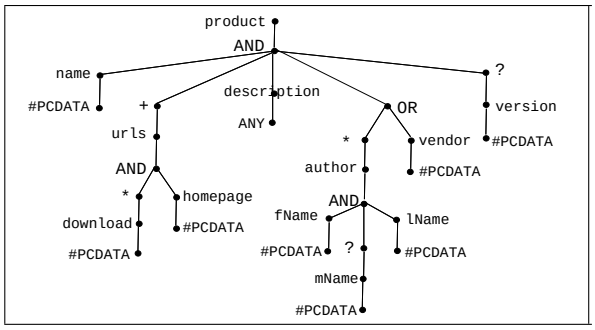

Figure 4: Tree representation of DTD of Fig. 2

the `?` operator represents an optional element, whereas the `*` and `+` operators represent repeatable elements (0 or more times, 1 or more times, respectively). In the matching process we do not consider sequences of unary operators (that is, `?`, `*`, `+`) because a concise and equivalent representation with a single operator always exists.

In our representation of DTDs each node corresponds to an element, or to an element type, or to an operator. In each tree representing a DTD the label of the root belongs to $\mathcal{EN}$ (it is the name of the main element of documents described by the DTD) and there is a single edge outgoing from the root. Moreover, there can be more than one edge outgoing from a node, only if the node is labeled by `AND` or `OR`. Finally, all nodes labeled by types are leaves of the tree. Let $\mathcal{ET}$ be the set of possible basic types for elements ($\mathcal{ET} = \{\#\texttt{PCDATA}, \texttt{ANY}\}$).

Definition 2 (*DTD*). A DTD is a labeled tree (T, φ_T) defined on the set of labels $\mathcal{EN} \cup \mathcal{ET} \cup \mathcal{OP}$ with the following properties:

1. T is of the form $(v, [T'])$ with $\varphi_T(v) \in \mathcal{EN}$;
2. for each subtree (v, C) of T, $\varphi_T(v) \in \mathcal{EN} \cup \mathcal{OP}$;
3. for each (leaf) subtree v of T, $\varphi_T(v) \in \mathcal{ET}$;
4. for each subtree (v, C) of T, if $\varphi_T(v) \in \{\texttt{OR}, \texttt{AND}\}$, then C=$[T_1, \ldots, T_n]$, $n > 1$;
5. for each subtree (v, C) of T, if $\varphi_T(v) \in \{?, *, +\} \cup \mathcal{EN}$, then $C = [T']$. □

We remark that the introduction of operators $\mathcal{OP} = \{\texttt{AND}, \texttt{OR}, ?, *, +\}$ allows us to represent the structure of all kinds of DTDs. The introduction of the `AND` operator is required in order to distinguish between an element containing an alternative between sequences (e.g. <!ELEMENT a(b|(c1,c2))>) and an element containing the alternative between all the elements in the sequence (e.g. <!ELEMENT a(b|c1|c2)>). The two different tree representations are shown in Figure 5(a,b). The document in Figure 5(c) is valid with respect to the DTD (b) but it is not valid with respect to the DTD (a).

3. Principles in Matching an XML Document against a DTD

In this section we introduce by means of some examples the behavior of the matching algorithm for the evaluation of the similarity between an XML document and a DTD. In particular we discuss the most relevant issues in this kind of match and how the algorithm addresses them.

We remark that we have chosen simple examples that allow us to focus on the behavior of the algorithm in common cases. The matching algorithm is complete enough to be used in the similarity evaluation of arbitrary documents and DTDs, characterized by any combination of the features discussed in this section.

3.1. Matching a Document against a set of Documents

Two different approaches can be devised for measuring the structural similarity between an XML document and a DTD: the DTD can be exploited either as a generator of document struc-

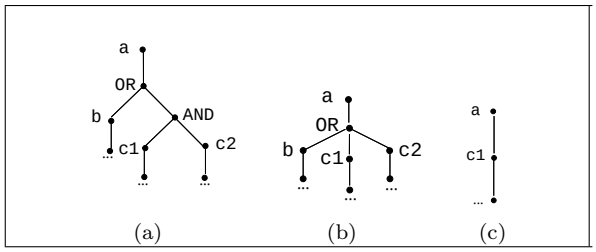

Figure 5: Example of DTDs motivating the introduction of AND labeled nodes

tures (*extensional approach*) or as a set of rules constraining the content of each element (*intensional approach*). According to the extensional approach, the set of possible document structures of documents valid for the DTD is considered.[3] By considering a document structure at a time, existing algorithms for measuring the structural similarity between XML documents [12,22] can be applied. The match resulting in the highest similarity value is considered as the best match and its evaluation as the structural similarity degree between the document and the DTD. According to the intensional approach, by contrast, the structural similarity measure is computed by means of a matching algorithm that compares the document structure to the DTD. The rules constraining the element contents are exploited for determining the best match. The set of document structures the DTD describes is not computed. Rather, the best structure for an element specification, for elements containing alternatives or repetitions, is locally determined as soon as the structure of its subelements in the document is known.

Since the extensional approach can result in exponential complexity even for very common cases, we present a matching algorithm based on the intensional approach. Note that also the intensional approach has, in the general case, exponential complexity. However, in a significant subset of cases, the most common in practice, the algorithm is polynomial as we show in Section 4.4.

[3]Note that this set can be infinite. Taking the document being matched against the DTD into account allows one to consider only a finite, though potentially big, set of document structures.

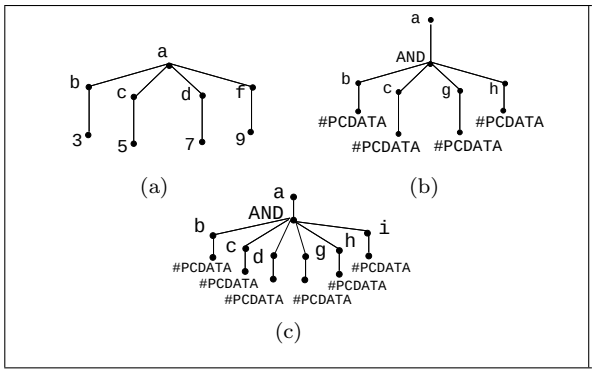

Figure 6: Identification of plus, minus and common elements

3.2. Common, Plus, and Minus Elements

The matching algorithm relies on the identification and proper evaluation of: elements appearing both in the document and in the DTD, referred to as *common* elements; elements appearing in the document but not in the DTD, referred to as *plus* elements; elements appearing in the DTD but not in the document, referred to as *minus* elements.

Example 1 *Consider the document D in Figure 6(a) and DTDs T_b, T_c in Figure 6(b,c), respectively. The matching algorithm identifies that D and T_b have the same tag for the document element but some of the subelements are different. In particular, D and T_b share elements* b *and* c*, whereas D contains elements* d *and* f *not appearing in T_b, and T_b contains elements* g *and* h *not appearing in D. Thus, the algorithm detects that the two structures have two common elements, two minus elements, and two plus elements.*

Consider now the DTD T_c. The matching algorithm determines that D and T_c share elements b*,* c*, and* d*, whereas D contains element* f *not appearing in T_c, and T_c contains elements* g*,* h *and* i *not appearing in D. Thus, the two structures have three common elements, one plus element, and three minus elements.* ○

In the two examples the identified common, plus and minus elements have to be properly evaluated in order to identify the best DTD between T_b and T_c. Obviously, to achieve the best similarity, plus and minus elements should be minimized

and common elements should be maximized. If we consider the absence of an element equivalent to the presence of an additional element, D is more similar to T_c because they have more common elements. However, there are situations in which plus and minus elements cannot be considered equivalent. For this reason we introduce α and β, two real numbers greater than 0, that allow us to properly weight plus and minus elements as we will discuss in Section 4.

The evaluation of such elements is performed by taking into account two main factors. First, the matching algorithm assigns a weight according to the level in which common elements are detected in the hierarchical structure of the two tree representations. Elements at higher levels in the document structure are more relevant than subelements deeply nested in the document structure. Then, the evaluation takes into account the structure of plus and minus elements. Complex elements have a greater impact on the evaluation than simpler ones.

In the remainder of the section we discuss how the matching algorithm determines the number of levels of a document/DTD, and the function *Weight* used for determining the structural complexity of an element.

3.2.1. Level of an Element

The similarity measure catches the intuition that elements at a higher level in a document are more relevant than elements at a lower level.

Example 2 *Consider the documents and the DTD in Figure 7. Element* `f`*, subelement of element* `d`*, is missing in the document in Figure 7(b). By contrast, element* `b`*, subelement of element* `a`*, is missing in the document in Figure 7(c). The document in Figure 7(b) is more similar to the DTD in Figure 7(a).* ○

We thus introduce the notion of level of an element, related to the depth of the corresponding tree. Given a tree T, representing a document, the level of T is its depth as a tree, that is, the number of nodes along the longest maximal path (that is, a path from the root to a leaf) in T. By contrast, given a tree T, representing a DTD, its level is the number of nodes, not labeled

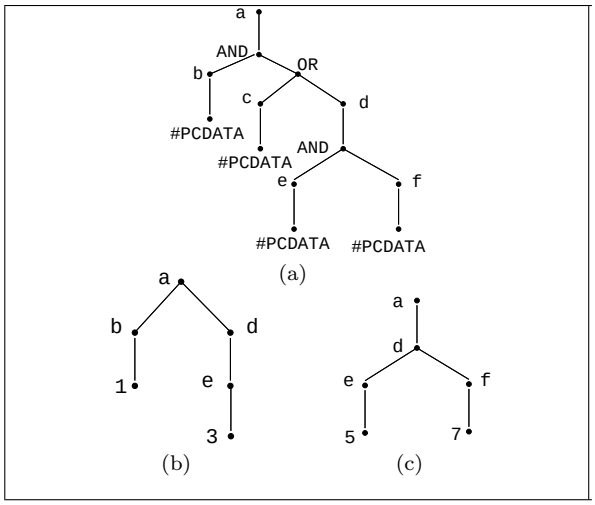

Figure 7: Documents and DTD of Example 2

by an operator, along the longest maximal path in T. This is because edges labeled by operators in DTD trees only influence the breadth of the corresponding document trees, not their depths. These notions are formalized by the following definition.

Definition 3 (*Function Level*). Let $T = (v, [T_1, \ldots, T_n])$ be a subtree of a document or a DTD. Function *Level* is defined as follows:

$$Level(T) = \begin{cases} 1 + max_{i=1}^{n} Level(T_i) & \text{if } \varphi(v) \in \mathcal{EN} \\ max_{i=1}^{n} Level(T_i) & \text{if } \varphi(v) \in \mathcal{OP} \\ 0 & \text{otherwise} \end{cases}$$ □

Example 3 *Let T denote the DTD in Figure 4, then $Level(T) = 3$.* ○

In computing the level of a tree leaves are not considered. This is because we are interested in the number of nested elements and leaves only have data contents. Now, the matching algorithm can assign a different weight to elements at different levels of the tree. Let $l = Level(T)$ be the level of a document/DTD T and γ be the factor of relevance of a level with respect to the underlying level, the root of T will have weight γ^l, and the weight is then divided by γ when going down a level to its children. Thus, for a generic level i of T, γ^{l-i} is the corresponding weight. Such weight is multiplied for the number of common,

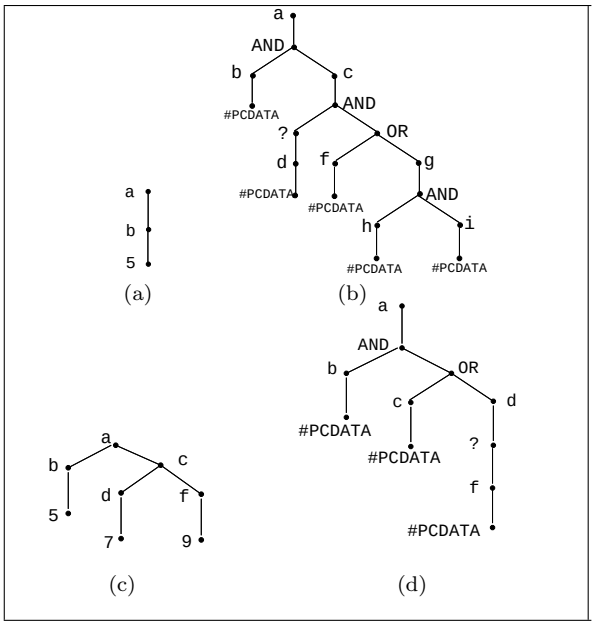

Figure 8: Documents and DTDs of Example 4

plus and minus elements identified at that level in order to take also the level into account in the match of the two structures.

3.2.2. Weight of an Element

In the evaluation of plus and minus elements the matching algorithm considers their structures, as shown in the following example.

Example 4 *Consider the documents and DTDs in Figure 8. If we match the document in Figure 8(a) against the DTD in Figure 8(d), we can see that the document lacks element* `c` *and the corresponding* `#PCDATA` *value. By contrast, if we match the document in Figure 8(a) against the DTD in Figure 8(b), we can see that the document lacks element* `c` *and the corresponding subtree. The lack of element* `c` *must be evaluated differently, since in the first case it has a simple data content, whereas in the second one it has a complex substructure. Consider now the document in Figure 8(c) and the DTD in Figure 8(d). The DTD specifies a* `#PCDATA` *content for element* `c`*, whereas in the document element* `c` *has a more complex substructure.* ○

The example above shows that the matching algorithm should take into account the structure of plus and minus elements. In case of minus elements, however, the structure is not fixed. Consider element `c` in Figure 8(b): it has an optional subelement (element tagged `d`) and an alternative of subelements (element tagged `f` or element tagged `g`). Our idea is to consider, as structure of the minus elements, the simplest document structure that can be generated from that portion of DTD. Thus, the measure should not take into account optional or repeatable elements and, in case of alternative elements, the measure should take into account only one of the alternative elements (reasonably the one with the simplest structure). We thus introduce function $\mathcal{W}eight$ to evaluate a subtree of a document or of a DTD.

Definition 4 (*Function* $\mathcal{W}eight$). Let T be a subtree of a document or a DTD (D, φ), and w_l be the weight associated with the level of T in D. Function $\mathcal{W}eight$ is defined as follows:[4]

$$\mathcal{W}eight(T, w_l) =$$

$$\begin{cases} w_l & \text{if } label(T) = \mathcal{V} \cup \mathcal{ET} \\ 0 & \text{if } label(T) \in \{*, ?\} \\ \mathcal{W}eight(T', w_l) & \text{if } label(T) = + \text{ and } T = (v, [T']) \\ \sum_{i=1}^{n} \mathcal{W}eight(T_i, w_l) & \text{if } label(T) = \texttt{AND} \text{ and } T = (v, [T_1, \ldots, T_n]) \\ min_{i=1}^{n} \mathcal{W}eight(T_i, w_l) & \text{if } label(T) = \texttt{OR} \text{ and } T = (v, [T_1, \ldots, T_n]) \\ \sum_{i=1}^{n} \mathcal{W}eight(T_i, \frac{w_l}{\gamma}) + w_l & \text{otherwise, where } T = (v, [T_1, \ldots, T_n]) \end{cases}$$ □

Given a subtree of the document and a weight w_l, function $\mathcal{W}eight$ multiplies the number of elements in each level for the weight associated with the level. The weight of the level is w_l for the first level, $w_l/2$ for the second level, $w_l/4$ for the third level, and so on. The resulting values are then summed. Given a subtree of the DTD and a weight w_l, function $\mathcal{W}eight$ works as on a document, but it takes into account only mandatory elements in the DTD. That is, the function does

[4] Given $T = v$ or $T = (v, C)$, $label(T) = \varphi(v)$.

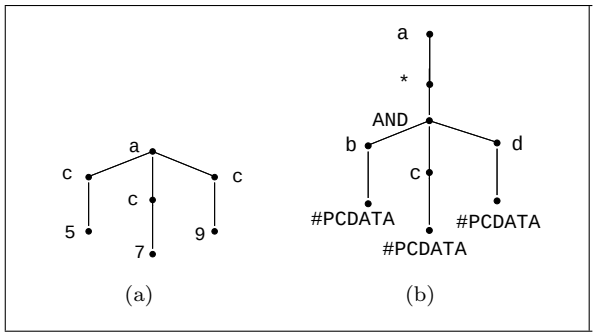

Figure 9: Document and DTD of Example 6

not consider optional elements or repeatable elements labeled by `*`. Moreover, in case of `OR` labeled nodes, the weights associated with the possible alternatives are evaluated and the minimal value is chosen. The choice of the minimal value corresponds to selecting the subtree with the simplest structure.

Example 5 *Let T be the DTD in Fig. 4, and assume $\gamma = 2$, $\mathcal{W}eight(T, 8) = 27$. Note that, in this case, the weight 8 is 2^3, where 3 is the number of levels of T. Moreover, the elements that contribute to the weight of T are the mandatory* `name`*, and* `description` *elements and the* `urls` *element because it is repeatable from 1 to many times (thus an occurrence is mandatory). The total weight of these elements is* 19*. The others do not contribute because they are optional or repeatable from* 0 *to many times. Note that, the* `OR` *subtree does not contribute to the weight because one of the alternatives it bounds is repeatable from* 0 *to many times. The weight of the root is* 8*, therefore the total weight is* 27*.* ○

3.3. Optional and Repeatable Elements

In case of repeatable elements, the similarity measure must identify the best number of repetitions, that is, the one that maximizes common elements and minimizes plus and minus elements. Note that a higher number of repetitions can result in every element in the document to match with an element in the DTD (no plus) but, by contrast, it can increase the number of unmatched elements in the DTD (minus). Optional elements can be considered as special cases of repeatable elements with a constraint on the maximal number of repetitions.

repetitions	common	plus	minus
0	0	3	0
1	1	2	2
2	2	1	4
3	3	0	6
4	3	0	9

Table 1
Measuring the similarity between the document and the DTD of Example 6

Example 6 *Consider the document D and the DTD T in Figure 9. The possibility of repeating an arbitrary number of times the sequence of elements (*`b`*,* `c`*,* `d`*) allows us to map each element* `c` *in D to a corresponding element in T. However, since D contains three* `c` *elements, the sequence in T must be repeated three times, resulting in a total of nine elements: three present in D (three* `c` *elements) and six missing from D (three* `d` *and three* `b` *elements). If, by contrast, we had repeated the sequence twice, we would have obtained two common elements and four minus elements. The situation is summarized in Table 1.* ○

The matching algorithm handles repeatable elements in the following way. The algorithm matches all the elements (at the current level) against the repeatable element in order to determine the evaluation of common, minus, and plus elements. After that, it determines the best number of repetitions by applying the evaluation function and choosing the maximal value. The evaluation is more complicated when a sequence or alternative of elements should be handled. The behavior of the algorithm in these situations is shown in the following section.

3.4. Sequences and Alternatives of Elements

The evaluation of sequences of elements is performed in two steps. The first step identifies the presence or absence of single elements of the sequence (i.e. it identifies the minus and common elements). Minus and common elements are evaluated as described above. Then, the sequence of

elements is evaluated by summing up the evaluations obtained for the single minus and common elements. The evaluation is more complicated when the sequence is repeatable. In this situation, indeed, the algorithm should identify the possible repetitions of the sequence. The evaluation of each sequence corresponds to the sum of the evaluation of common and missing elements and the best number of repetitions of the sequence is determined by exploiting the evaluation function.

Example 7 *Consider the document and the DTD of Example 6. In the evaluation of the* `AND` *operator (which is repeatable), the algorithm first finds that element* `c` *in the DTD has 3 matches in the document, whereas elements* `b` *and* `d` *in the DTD have no match in the document. Taking into account the evaluation obtained for the single elements, the possible repetitions of the sequence are computed. Zero repetitions of the sequence means that the three* `c` *elements are plus, and there are no common and minus elements. One repetition of the sequence means that one of the three* `c` *elements is common, the other two* `c` *elements are plus, and* `b` *and* `c` *elements are minus. The other repetitions of the sequence are computed in a similar way. Note that, a new repetition of the sequence is considered till a common element has to be matched. Therefore, in this case four repetitions are considered. The best one is then selected by means of the evaluation function.* ○

The matching algorithm handles alternatives in a similar way. However, in this case the evaluations obtained are not summed up, rather the best one, that is the evaluation corresponding to the best alternative among the possible ones, is chosen.

3.5. Role and Setting of Parameters

The behavior and results of the matching algorithm rely on some parameters previously outlined and reported in Table 2.

A user sets these parameters depending on the application domain in which the matching algorithm is used. Some examples will be shown in Section 5 when we discuss some applications of the matching algorithm.

Parameter	Description
α	weight of plus elements ($\alpha \geq 0$)
β	weight of minus elements ($\beta \geq 0$)
γ	relevance factor of a level ($\gamma \in \mathbb{N}$)

Table 2
Parameters of the matching algorithm

Depending on the values assigned to α and β, the matching algorithm gives more relevance to plus elements with respect to minus elements, or vice-versa. For example, if $\alpha = 0$ and $\beta = 1$ plus elements are not taken into account in measuring similarity. Therefore, a document with only extra elements with respect to the ones specified in the DTD has a similarity degree equal to 1. By contrast, if $\alpha = 1$ and $\beta = 0$ the minus elements are not taken into account in the similarity measure. In the following examples we assume that $\alpha = \beta = 1$, thus giving the same relevance to plus and minus elements.

Depending on the value assigned to $\gamma \in \mathbb{N}$, the matching algorithm gives more relevance to common elements at higher levels in the document with respect to others at lower levels. By taking $\gamma = 1$ all the information is considered equally relevant, and thus the fact that elements appear at different levels in the nested structure is not taken into account. By contrast, taking $\gamma = 2$ elements at a given level have double relevance with respect to their children. In what follows, we consider $\gamma = 2$.

4. The Matching Algorithm

In the previous section we have outlined the behavior of the matching algorithm in the most relevant cases. In this section we point out some details of the developed algorithm.

4.1. Evaluation Function

In order to obtain the best match between the two structures, common elements must be maximized, whereas plus and minus elements must be minimized. However, we want to obtain a numeric value that quantifies the similarity between

the document and the DTD. Thus, we assume plus, minus, and common elements to be evaluated to three natural values p, m, c, taking into account the levels and the weights, as discussed in Section 3. These three values are combined through an *evaluation function* for determining an overall similarity evaluation. The evaluation function we choose is function $\mathcal{E}$, formally defined in the following, which is based on the *ratio model* [30]. This function computes the ratio between the evaluation c of the common elements between the two structures (i.e., elements in the "intersection" between the two structures) and the evaluation $p+m+c$ of all the elements in the two structures (i.e., elements in the "union" of the two structures). The evaluation of plus and minus elements are weighted according to α and β parameters. The obtained similarity value is a real number in the range [0,1].

Definition 5 (*Function* $\mathcal{E}$). Let (p, m, c) be a triple of natural numbers and α, β be real numbers s.t. $\alpha, \beta \geq 0$. Function $\mathcal{E}$ is defined as:

$$\mathcal{E}(p,m,c) = \begin{cases} 0 & \text{if } (p,m,c) = (0,0,0) \\ \frac{c}{\alpha p + c + \beta m} & \text{otherwise} \end{cases} \quad \square$$

Relying on function $\mathcal{E}$, an order relationship $\preceq$ has been defined. This order is exploited for selecting among a set of matches (represented as (p, m, c) triples) the optimal ones (i.e. the maximal triple). Details on the $\preceq$ order can be found in [6].

Example 8 *Consider the document and the DTD in Figure 10. Since the document only contains element* `b1`*, if we choose the right branch of the* `OR` *we have one common element and 39 missing elements. By contrast, if we choose the left branch of the* `OR`*, we have no common elements, but only a plus and a minus element.* ○

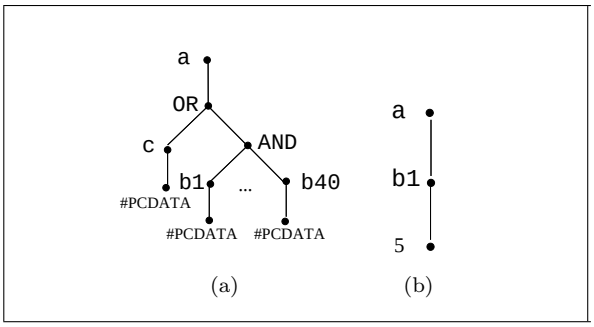

Figure 10: Tree representations of document and DTD of Example 8

4.2. A Sketch of the Matching Algorithm

An algorithm, named $\mathcal{M}atch$, that allows one to assign a (p, m, c) triple to a pair of trees $(document, DTD)$ has been defined. Such algorithm is based on the idea of locally determining the best structure for a DTD element, for elements containing alternatives or repetitions, as soon as the information on the structure of its subelements in the document are known.

The algorithm is general enough to evaluate the similarity between any kind of XML documents and DTDs. In this paper, however, we focus on the most meaningful *core* of the algorithm, based on the assumption that, in the declaration of an element, two subelements with the same tag are forbidden. That is, element declarations such as `<!ELEMENT a (b*, (c|b))>` are not considered. Details on the general version of the algorithm can be found in [6,21].

Given a document D, and a DTD T, algorithm $\mathcal{M}atch$ first checks whether the root labels of the two trees are equal. If not, then the two structures do not have common parts, and a null triple is returned. If the root labels are equal, the maximal level l between the levels of the two structures is determined, and the recursive function $\mathcal{M}$ is called on:

1. the root of the document,
2. the first (and only) child of the DTD,
3. the level weight (γ^{l-1}) taking into account that function $\mathcal{M}$ is called on the second level of the DTD structure,
4. a flag indicating that the current element (the root element) is not repeatable.

Function $\mathcal{M}$ recursively visits the document and the DTD, at the same time, from the root to the leaves, to match common elements. Specifically, two distinct phases can be distinguished:

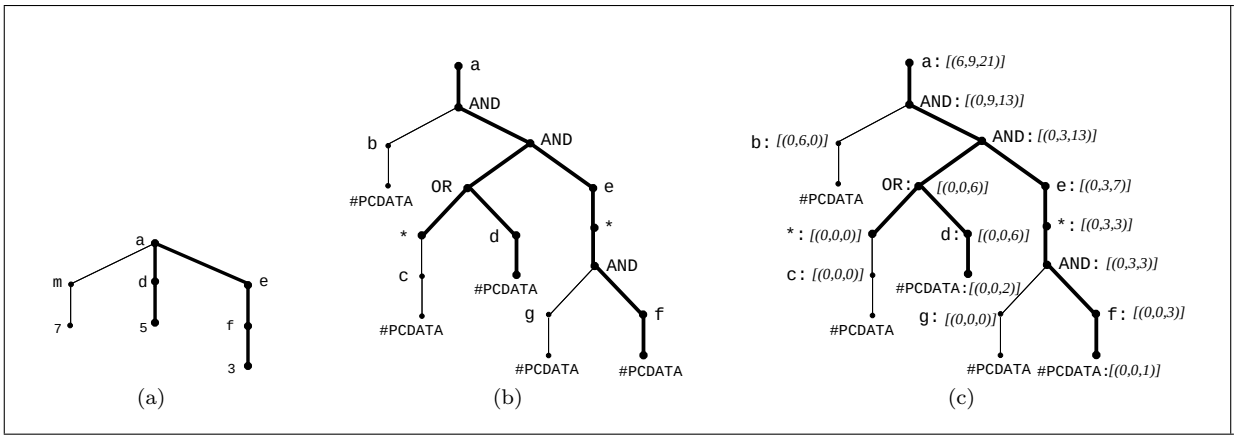

Figure 11: Execution of function $\mathcal{M}$

1. in the first phase, moving down in the trees from the roots, the parts of the trees to visit through recursive calls are determined, but no evaluation is performed;

2. when a terminal case is reached, on return from the recursive calls and going up in the trees, the various alternatives are evaluated and the best one is selected.

Intuitively, in the first phase the DTD is used as a "guide" to detect the common elements between the document and the DTD, disregarding the operators that bind together subelements of an element. In the second phase, by contrast, the DTD operators are considered in order to verify which elements are bound as prescribed by the DTD, and to define an evaluation of the missing or exceeding parts of the document with respect to the DTD. Terminal cases are the following: a leaf of the DTD is reached, or an element of the DTD not present in the document is found. In these cases a (p, m, c) triple is returned. Then, the second phase starts and the evaluation of internal nodes is performed, driven by their labels.

4.3. An Illustrative Example

We now illustrate the behavior of function $\mathcal{M}$ on the document and the DTD in Figure 11(a,b). For sake of clarity, in the discussion of the algorithm, we denote the element of the document labeled by `a` as $\mathtt{a}_D$, and the element of the DTD labeled by `a` as $\mathtt{a}_T$.

During the first phase, function $\mathcal{M}$, driven by the label of the current DTD node, is called on subtrees of the document and the DTD. For example, on the first call of $\mathcal{M}$ on ($\mathtt{a}_D$, $\mathtt{AND}_T$), recursive calls on $\mathtt{a}_D$ and all the subtrees of $\mathtt{AND}_T$ are performed (i.e., on ($\mathtt{a}_D$, $\mathtt{b}_T$), and ($\mathtt{a}_D$, $\mathtt{AND}_T$)). Recursive calls are performed disregarding the operators in the DTD and moving down only when an element declared in the DTD is found in the document as child of the current node. Moreover, in such cases, the weight level is divided by γ in order to determine the level weight of the underlying level. Figure 11(a,b) shows the performed recursive calls. An edge (v, v') of the tree is bold if a recursive call of function $\mathcal{M}$ has been made on the subtree rooted at v'. Note that no recursive calls have been made on $\mathtt{b}_T$, $\mathtt{c}_T$, and $\mathtt{g}_T$ because such elements are missing in the document. Note also that $\mathtt{m}_D$ has not been visited by function $\mathcal{M}$, because this element is not required in the DTD.

When a terminal case is reached, a (p, m, c) triple is produced. For example, when function $\mathcal{M}$ is called on ($\mathtt{f}_D$, $\mathtt{\#PCDATA}_T$), the triple $(0, 0, 1)$ is generated, because the DTD requires a data content for $\mathtt{f}_D$ and, actually, such element has a textual content. By contrast, when function $\mathcal{M}$ is called on ($\mathtt{a}_D$, $\mathtt{b}_T$), the triple $(0, 6, 0)$ is generated, because the DTD requires an element tagged `b`, but such element is missing in the document. Therefore, function *Weight* is called on $\mathtt{b}_T$ and, since the current level weight is 4, the value 6 is returned as weight of the missing subtree.

On return from the recursive calls, the operators and the repeatability of the node are considered in order to select the best choice among the possible ones for binding together subelements. For example, returning from the evaluation of subtrees of the `OR` element, which is not repeatable, the triples $(0,0,0)$ and $(0,0,6)$ obtained for the evaluation of subtrees are considered. The best one is selected relying on the $\mathcal{E}$ evaluation function. By contrast, returning from the evaluation of subtrees of an `AND` element, which is not repeatable, the obtained evaluations are summed in order to determine the evaluation of the sequence of elements. The behavior of the algorithm is much more articulated when elements are repeatable. In such cases, indeed, not only a triple is generated, but a list of triples. The lists of triples are then combined in order to evaluate internal nodes.

The intermediate evaluations are reported in Figure 11(c). If an edge is bold the label is followed by the (p,m,c) triple obtained from the evaluation of the corresponding subtree. If an edge is not bold, but the label is followed by a (p,m,c) triple, it represents the evaluation of minus elements of the subtree.

The triple associated with the main element of the DTD (i.e. $(6,9,21)$) is obtained by the $\mathcal{M}atch$ algorithm summing up the evaluation returned by function $\mathcal{M}$ ($(0,9,13)$), the evaluation of the plus element `m` ($(6,0,0)$) and the identification of common root label ($(0,0,8)$).

4.4. Algorithm Complexity

The running time of the $\mathcal{M}atch$ algorithm depends on the running time of function $\mathcal{M}$. Let M be the number of nodes of the document, N be the number of nodes of the DTD, and Γ the maximal number of edges outcoming from a node of the document, the running time of function $\mathcal{M}$ is $\mathcal{O}(\Gamma^2 \cdot (N+M))$ [6]. This complexity deeply depends on the assumption we started with. That is, the assumption that in the declaration of an element, two subelements with the same tag are forbidden.

Relying on this assumption, in the first phase of the algorithm, recursive calls are performed only until common elements between the structures are detected. In this phase of the algorithm no "wrong matches" are determined, because at most one match is possible between an element of the document and an element of the DTD. Therefore, this phase has a running time linear in the number of nodes of the two structures. Then, in the second phase, the matching algorithm evaluates the DTD operators. For each common element, this phase has a running time quadratic in the number of edges outcoming from the node of the document. Combining the two results, the above complexity is obtained.

In the general version of the algorithm [6] the above assumption has been removed. In such a case, "wrong matches" can arise during the first phase. For example, consider an element of the document that matches with n elements with the same tag in the DTD. In order to identify the best match, the second phase should be performed n times. Each time the element of the document is considered in common with one of the n elements of the DTD, and, at the end, the match that maximizes the evaluation function is chosen. It is easy to understand that, when the numbers of elements with the same tag either in the document or in the DTD increases, the complexity of the general version of function $\mathcal{M}$ changes from polynomial to exponential.

We would like to remark, however, that the presence in the DTD of elements with the same tag is often due to a wrong design of the DTD. However, in [6] some techniques have been proposed for reducing the execution time of the matching algorithm, even if, in the worst case, the complexity is still exponential.

4.5. Similarity Measure

The similarity measure between a document and a DTD is defined as follows.

Definition 6 (*Similarity Measure*). Let D be a document and T a DTD. The similarity measure between D and T is defined as follows:

$$\mathcal{S}(D,T) = E(\mathcal{M}atch\langle D,T\rangle) \qquad \square$$

Example 9 *Let D and T be the document and the DTD in Figure 11(a,b). Their similarity degree is* $S(D,T) = \mathcal{E}(\mathcal{M}atch\langle D,T\rangle) = \mathcal{E}(\langle 6,9,21\rangle)$=0.58. $\bigcirc$

The following proposition states the relationship between the notion of validity and our similarity measure.

Proposition 1 *Let D be a document, T a DTD, and α, β the parameters of function $\mathcal{E}$. If $\alpha, \beta \neq 0$ the following properties hold:*

- *If D is valid with respect to T, then $\mathcal{S}(D,T) = 1$;*
- *If $\mathcal{S}(D,T) = 1$, then D is valid with respect to T, disregarding the order of elements.* △

Sketch of Proof of Proposition 1. The first assertion follows from the fact that if a document D is valid for a DTD T, this means that its structure is exactly one of the structures described by the DTD. Thus, the document neither contains elements not appearing in the DTD (thus, plus = 0), nor it misses elements required by the DTD (thus, minus = 0). Therefore, when function $\mathcal{E}$ is applied, the ratio between c and $0+0+c$ is computed, thus obtaining 1. The second assertion holds since the similarity value can be 1 only if the two values of which we compute the ratio are equal. Since α and β are not null, and the p, m, c values are natural, thus, non negative, this can happen only if $p = m = 0$. This means that the document neither contains elements not appearing in the DTD, nor it misses elements required by the DTD. Thus, according to the notion of validity, if we disregard the order of elements, the document is valid for the DTD. △

5. Applications

In this section we discuss applications of the matching algorithm we are investigating.

5.1. Classification of Documents

A first application of the matching algorithm is for the classification of XML documents gathered from the Web against a set of DTDs declared in an XML database. The scenario we refer to is characterized by a number of heterogeneous databases of XML documents able to exchange documents among each other. Each database stores and indexes the local documents according to a set of local DTDs. An XML document entering a database is matched, by means of the matching algorithm, against the local DTDs. If a DTD exists to which the document conforms according to the usual notion, then the document is accepted as valid for this DTD. Otherwise, the proposed algorithm is used for selecting the DTD, among the ones in the database, that best describes the structure of the document. In this scenario, a *similarity threshold* should be fixed. Such threshold represents the minimal degree of similarity required for binding an XML document to a DTD. Obviously, the DTD for which the similarity degree is the highest, and above the fixed threshold, is selected. Whenever the similarity degree is not above the threshold for any DTD of the database, the document is considered "unclassified" and stored in a repository of unclassified documents. For the retrieval, protection and indexing of such documents none of the facilities specified at DTD level can be applied.

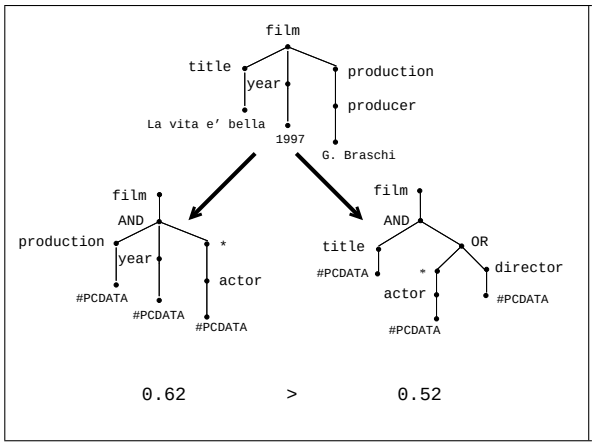

Figure 12: Classification of a document

Example 10 *Consider the document D and the two DTDs T_1 and T_2 in Figure 12. The similarity degree between D and T_1 is $\mathcal{S}(D,T_1) = 0.62$, whereas the similarity degree between D and T_2 is $\mathcal{S}(D,T_2) = 0.52$. Document D is more similar to DTD T_1 than to T_2, because $\mathcal{S}(D,T_1) > S(D,T_2)$.*

If we set the similarity threshold to 0.6, document D is classified in T_1. By contrast, if we set

the similarity threshold to 0.8*, document D cannot be classified in* T_1*, and, thus, it is stored in the repository of unclassified documents.* ○

Several experiments have been carried on in order to assess the similarity measure and the matching algorithm both from the correctness and from the efficiency viewpoint. First, we considered both real and synthetic data and classified them against a set of DTDs in order to verify that the algorithm correctly ranks documents according to the similarity measure. In both the experiments, we obtained that for each document D, and for each pair of DTDs T_1, T_2 such that D is not valid neither for T_1 nor for T_2, whenever $\mathcal{S}(D, T_1) > S(D, T_2)$, D actually is more similar to T_1 than to T_2 [6]. Then, some performance evaluations have been carried on in order to show that the matching algorithm is reasonably efficient to be used in practice. Note that this is a crucial issue as similarity checks are supposed to be performed frequently and online. The execution time of the algorithm varies from few milliseconds for simple XML documents and DTDs, to few seconds for very huge documents and DTDs (i.e. whose size is in the order of 4-5 Mbytes).

5.2. Evolution of DTD Structures

After having classified a certain number of documents, the documents instances of a DTD can present some regularities that, if captured by the DTD, would restrict the divergence between the structure of documents as specified by the DTD and the actual structures of documents instances of the DTD. The goal of the evolution approach is to capture these regularities thus adapting the set of DTDs to the set of documents. Preliminary results have been reported in [7].

The data flow of the evolution approach is shown in Figure 13, in which rectangles denote the main functional components of the approach, cylinders denote data stores, thick arrows denote the control flow, and thin arrows denote data flow. Each time a document, created outside the database, enters the database it is initially inserted in a queue of "to-be-processed" documents. When it is then selected, it is associated with a DTD of the database, that is, the one best describing its structure, through the classification algorithm. If a document, matched against each DTD, does not produce a similarity value above the similarity threshold, it is inserted in the repository of unclassified documents. Otherwise, the document is handled as an instance of the DTD for which the evaluation produced the highest similarity value.

Once the classification phase is completed (i.e, the DTD of which the document is an instance has been selected) some structural information are extracted from the document. Specifically, information about frequent "*patterns*" identified in the elements of a document that are not valid w.r.t the corresponding DTD declaration. A pattern is a subset of the tags of subelements of a non-valid element e_{nv} of the document with respect to a DTD. Patterns are used for identifying groups of subelements of e_{nv} frequently bound together and, thus, to extract the new structure of the DTD declaration of e_{nv}. In the recording phase this information is associated with the DTD in a data structure referred to as *extended DTD*. The use of this information avoids analyzing again the document in the subsequent phases. Moreover, this information is structural rather than content information, and it is aggregate over the whole set of analyzed documents, and thus it does not require much storage space. These activities are iterated till the evolution phase is triggered.

The *evolution phase* is activated after a certain number of documents have been classified. The evolution phase has a high cost in terms of rewriting the applications that are working on the database. Therefore, it should be triggered whenever the DTD is not representative anymore of its instances and such "update" improves the performance of applications that work on them. The event can depend on the access frequency to the DTD instances, on the number of non-conforming elements w.r.t the DTD, and on the number of documents currently considered as instances of the DTD. The *check component* is responsible to determine whether the evolution phase should be activated.

The *evolution phase* of the evolution process is responsible for generating a new set of DTDs

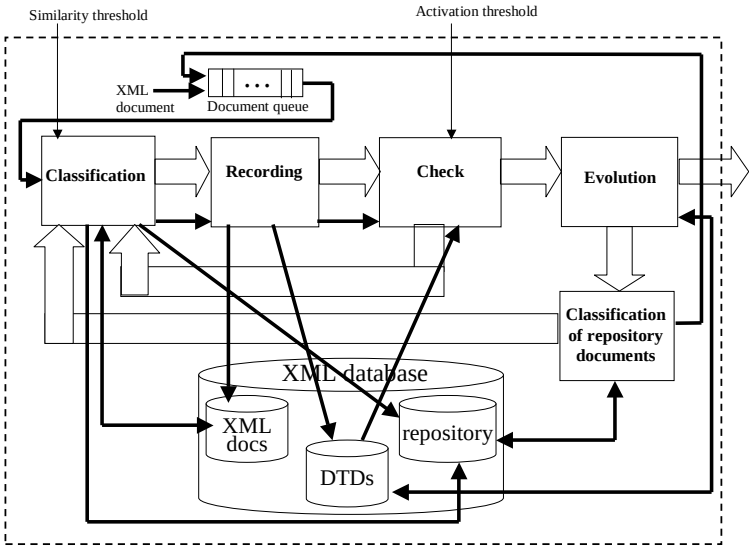

Figure 13: Data flow of the evolution approach

and can work at different granularities, ranging from a very coarse granularity, regenerating the whole DTD, to a very fine granularity, regenerating the structure of a single element in the DTD. By making use of the information collected in the recording phase, some association rules are extracted that represent relationships between presence/absence of subelements of an element. Based on such rules and on some heuristics we have identified, the new DTD is generated. Finally, after the evolution phase, the documents in the repository are classified again against the restructured set of DTDs in order to check whether the similarity is now above the threshold for a DTD of the database so that the document can be considered as instance of such a DTD.

The evolution phase is based on three key principles.

1. *Use of data mining association rules [17,18] for determining the most frequent patterns in the structure of subelements of each element.* For each element of the DTD, by relying on the patterns stored in the data structure, it is possible to determine elements that are always together (i.e. bound by an `AND` operator), elements that are never together (i.e. bound by an `OR` operator), elements, or groups of elements, that are repeated the same number of times (i.e. bound by a `*` or `+` operator), elements, or groups of elements, that are optional (i.e. bound by an `?` operator).[5] Moreover, in order to establish when the presence of an element implies the absence of another element, association rules like "*if element* `a` *is absent then element* `b` *is present*" have been considered.

2. *Incremental modification of the DTD.* Approaches proposed in [13,16] for inferring the "type" of a set of documents consider all the documents at once. Therefore, when a new documents is added to the set, in order to determine a new "type", the process starts from scratch. By contrast, in our approach we incrementally store the relevant information in the data structures and use them during the evolution process.

3. *Relevance of previous instances of the DTD.* Different relevance can be given to the current structure of the DTD with respect to

[5]Note that the terms "always", "never", and "same number" should be considered in their statistical sense, i.e. in *most cases*.

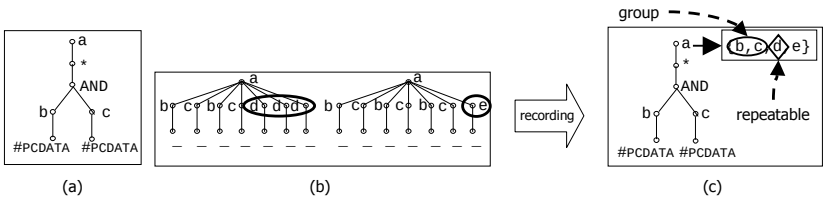

Figure 14: (a) DTD, (b) kind of documents classified against the DTD, (c) extended DTD

the documents classified against it since last DTD evolution. If the DTD was a *dummy* DTD generated from a training set of documents or, for the particular application area, the rule "more recent, more relevant" holds, then the DTD evolution process should forget the previous structure of the DTD and deeply modify it in order to obtain a new structure that closely represents the documents classified in the DTD since last evolution. By contrast, if the DTD structure is consolidated we want to minimize the DTD modifications in order to cover both the previous structure of the documents and the new structure deduced from the document classified since last evolution.

Example 11 *Let* `T` *be the DTD in Figure 14(a) and* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *be two sets of documents whose structures are reported in Figure 14(b). The label of root elements is* `a` *both for documents in* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *and all documents contain a sequence of* `b` *and* `c` *elements. However, this sequence in documents in* $\mathcal{D}_1$ *is followed by a sequence of* `d` *elements, whereas in documents in* $\mathcal{D}_2$ *it is followed by an* `e` *element. Documents in both sets are not valid with respect to T. Figure 14(c) presents a sketch of the extended DTD. Element* `a` *is associated with the set* $\{\texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}\}$ *of element tags found in the documents classified against T. Moreover,* $\{\texttt{b}, \texttt{c}\}$ *forms a group since elements* `b` *and* `c` *are repeated the same number of times and element* `d` *is marked as repeatable and optional (some documents do not contain it).*

Suppose that according to the "more recent, more relevant" rule, we decide to update the DTD structure. The evolution algorithm, by means of a set of policies, determines the new structure of the DTD. We do not detail the heuristic policies developed and simply outline the behavior of the algorithm in this specific example by means of Figure 15.

The evolution algorithm first determines that elements `b` *and* `c` *appear always together (i.e., the presence of* `b` *implies the presence of* `c`*, and vice versa), and they have the same number of occurrences (i.e., they form a group). Therefore, the new tree (1) in Figure 15 is obtained. Then, the evolution algorithm determines that elements* `d` *and* `e` *are complementary (i.e., the presence of* `d` *implies the absence of* `e`*, and the absence of* `e` *implies the presence of* `d`*), and* `d` *is repeatable. Therefore, the new tree (2) in Figure 15 is obtained. Trees (1) and (2) in Figure 15 are, finally, combined together (tree (3) in Figure 15) by means of the* `AND` *operator in order to obtain the final new DTD structure reported as tree (4) in the figure.* ○

5.3. Structural Queries

Recent approaches to the retrieval of XML documents exploit the structure of documents for improving both accuracy and efficiency. Such queries are referred to as *structural queries*. Moreover, several of those approaches have the capability of returning ranked answers, in the spirit of Information Retrieval.

Structural queries are normally expressed as labeled trees, representing either structural or content constraints on the documents which are possible answers to the query. By means of a match between the tree representation of the structural query and an XML document it is possible to verify whether the document is an answer to the query, to compute their degree of similarity, and to extract the parts of the document that the query should return.

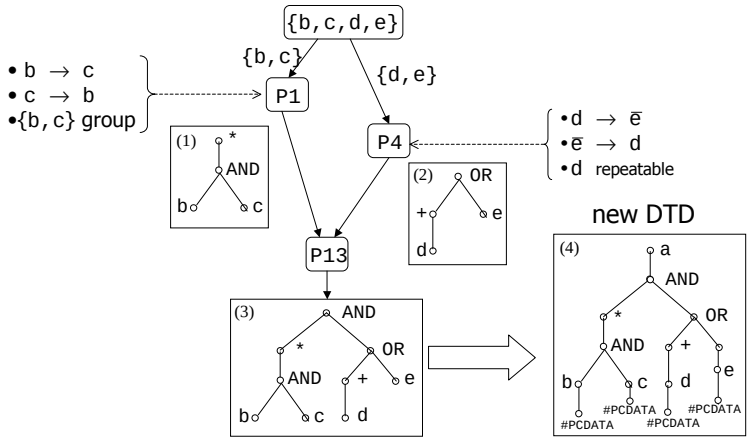

Figure 15: Application of the evolution algorithm

In our context, a structural query can be represented as a DTD, in which some additional constraints on the value of data content elements have been posed.[6] Therefore, a query is modeled as a labeled tree representing the structural and content constraints a document should verify in order to be considered an answer to the query. Then, the matching algorithm can be exploited for evaluating the degree of similarity between the two structures. If such a degree is above a given threshold, the document is added to the set of answers for the query (*query answer set*). The query answer set is ranked relying on the similarity degree.

Two different interpretations can be given to a structural query expressed as a labeled tree. First, it can represent a *template* of the documents we are looking for. Second, it can represent the minimal constraints a document should meet in order to belong to the query answer set. According to this interpretation, a document can contain other elements with respect to those of the query in which some conditions have been specified.

With a small extension of the matching algorithm we developed (in order to handle content conditions) both the interpretations are supported. The first one is obtained without any effort. Actually, this corresponds to the application of the classification approach in which some additional constraints have been added for the data content elements. By contrast, the second one is obtained by setting α to 0. In this way all the plus elements found in the document are not considered in the evaluation. Therefore, only elements required by the query but not present in the document are taken into account for properly evaluating the similarity degree between the document and the query.

Example 12 *Consider the following query expressed through the Xpath [33] notation:*

/film[director="Fellini"]
AND
/film[date > "1974"]

By exploiting the structure of the document deduced by the query formulation, the tree representation in Figure 16 can be generated.

Consider now the documents in the lower side of Figure 16. Assuming the interpretation of a query as a document template the similarity degrees the matching algorithm returns are used to rank the documents. ○

Some experiments have been carried on for testing the approach. The obtained results are

[6] Note that by exploiting the DTD operators it is possible to express structural queries more powerful than the ones possible with current approaches. The ? operator, for example, can be exploited for expressing optional conditions.

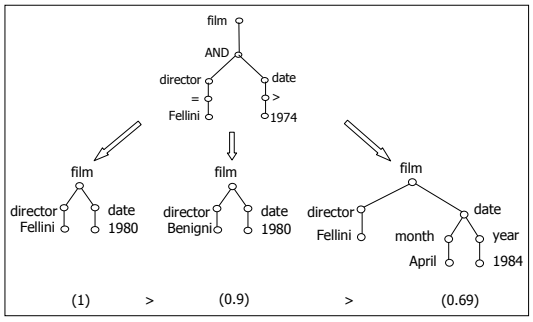

Figure 16: Evaluation of a structural query

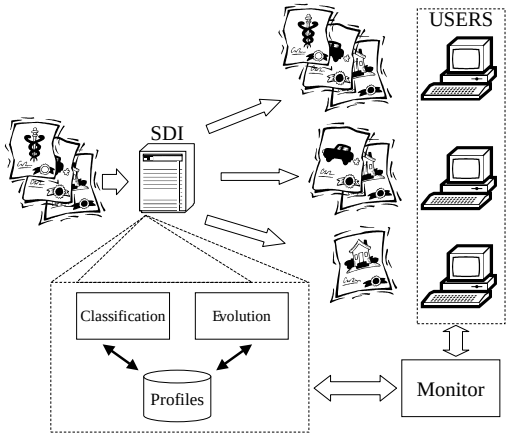

Figure 17: Integration of the classification and evolution approaches in a SDI

similar to those for the classification of XML documents against a set of DTDs.

5.4. Selective Dissemination of XML Documents

As the amount of XML data available online and the number of pervasive applications that take advantages of these data increase, systems that support *selective dissemination of information* (called *SDI systems*) are more and more popular [29].

A selective dissemination system manages user preferences as well as a stream of incoming documents. For each incoming document, the system searches for the set of user preferences that match it in order to identify the users to whom the document should be broadcasted. Users can set their preferences when they connect the first time to the system (by filling up a form) or the preferences can be dynamically discovered by monitoring the documents users frequently access. A key capability of a SDI system is the effective filtering of a continuous stream of XML documents according to user preferences. Indeed, a too selective filter may not send any documents to users, while a too liberal filter may spam users with irrelevant documents. Another key capability of a SDI system is the adaptability of user profiles to the new user preferences. In a dynamic world as the Web, it is not possible to assume that the preferences of users do not change.

Our classification and evolution approaches can be employed for enhancing a SDI system in order to provide such key capabilities. Specifically, our classification approach can be used to filter XML documents based on their structure and content. A user profile could be expressed as a DTD, in which some constraints on the value of data content elements have been posed. This DTD could initially be specified by the user or automatically inferred from documents previously deemed valuable by the user, by means of document clustering [22,29] and structure extraction techniques [13]. A selective dissemination of information can then be implemented by matching each document in the continuous incoming data stream against the DTD(s) modeling the user profile, and distributing the document to a user if it is similar enough, according to our measure, to her profile. Moreover, our evolution approach can be exploited for automatically adapting a set of DTDs to the actual set of documents classified against the user profiles. The evolution of the user profiles allows users to receive documents they actually are interested in. During the classification of the incoming documents against the user profiles, some structural and content properties can be extracted and used for the evolution of the profiles to the new preferences the users express by accessing such documents.

Figure 17 shows how our classification and evolution approaches can be integrated in a SDI system. A SDI system receives a continuous

stream of XML documents and, by classifying them against the user profiles, filters out the users that are not interested in the documents. Then, documents are broadcasted only to the interested users. A *Monitor* can check the documents users normally access in order to generate the initial user profiles and also the documents, among those broadcasted to a user, that she actually accesses. In this way, a weight can be associated with the documents actually accessed and used during the execution of the evolution process in order to give more relevance to these documents.

5.5. Protection of XML Documents

XML security has become a relevant research topic due to the widespread use of XML as the language for information representation and exchange on the Web. In [3] a suitable access control model has been defined that addresses all the fundamental requirements for access protection of XML documents: varying granularity levels of protection ranging from a single element of a document to a set of documents; the presence into a database of both valid and well-formed documents; hierarchical inter-linked structure of XML documents; releasing of documents relying on subject properties (specified by means of credentials) instead of subject identity. Moreover, a prototype, called Author-$\mathcal{X}$ [4], has been developed. Author-$\mathcal{X}$ provides facilities for enforcing the security policies and for helping the Security Officer in his work to protect the sensitivity information contained in huge amount of documents gathered from the Web.

In this field the matching algorithm has been exploited for the protection of new, well-formed XML documents that enter the database. Indeed, an important issue is the definition of access control policies to such documents. Because it is most likely the case that authorization policies are specified in terms of DTDs (i.e. at schema level), it is important to discover whether a well-formed document conforms to an existing DTD. In such a case, the document can be totally or partially covered by the policies defined for this DTD, and a DTD-based policy can be adopted. Otherwise, only document-based policies can be adopted to specify all required authorizations directly on the well-formed document.

The use of the matching algorithm has two main advantages. First, the matching algorithm is used for the classification of the documents against the set of DTDs. Then, it is used for identifying the parts of the document that are covered (or not) by the DTD. The DTD policies are directly propagated to the covered parts. Moreover, some *propagating options* can be exploited for associating an authorization policy to the parts of the document that are not covered by the DTD.

Example 13 *Suppose to have an XML document that is valid with respect to the DTD in Figure 2 with the exception of element* `author`*. Element* `author` *is not valid because it contains the subelement* `affiliation` *which is not required by the DTD. According to the* downward propagation option*, if an access control policy has been specified for the* `author` *element, such policy can be propagated to the* `affiliation` *subelement.* ○

In this context, then, the matching algorithm is used for identifying the DTD that best covers a document. Therefore it is more relevant to minimize the number of plus elements rather than the minus one. This behavior can be enforced by setting the β parameter to zero and, thus, disregarding minus elements and giving more relevance to plus elements in the evaluation of the structural similarity.

The presence of the matching algorithm and of the propagation options offer a different degree of automation and support to the Security Officer. The Security Officer can exploit such facilities for automatically protecting the content of documents entering the database, or for identifying the policies that could be associated with the new documents and then decide whether the automatic policies should be enforced, or it is better to enter new specific policies.

6. Related Work

In this section we review related work for measuring the structural similarity at document layer, at schema layer, and between the two layers, and compare the different approaches with ours.

6.1. Structural Similarity among Documents

Some approaches have been proposed [12,22] to quantify the structural similarity between XML documents. The main application of these proposals is for structural clustering. As clustering [26] assembles together documents with similar terms, structural clustering assembles together documents with a similar structure. Two relevant fields of application of structural clustering are the integration of semi-structured data and structural analysis of Web sites [12]. Indeed, grouping structurally similar documents can help in recognizing sources containing the same kind of information and in presenting the information provided by a site. Both approaches only focus on the structure of the documents, disregarding their content, that is, values of data content elements.

In [22] Nierman and Jagadish measure the structural similarity among XML documents. Since XML documents are modeled as ordered labeled trees, they suggest to measure the distance between two ordered labeled trees relying on the notion of tree edit distance [27,34,35]. However, two XML documents produced from the same DTD can have very different sizes due to optional and repeatable elements. Any tree edit distance that permits changes to only one node at a time will necessarily find a large distance between such a pair of documents, and consequently will not recognize that these documents should be clustered together as being derived by the same DTD. Thus, they develop an edit distance metric that is more indicative of this notion of structural similarity. Specifically, in addition to insert, delete, and relabel operations, they also introduce the insert subtree and delete subtree editing operations, allowing the cutting and pasting of whole sections of a document. Then, both for computational reasons and for improving their results in the XML domain, they restrict themselves to "allowable sequences" of edit operations.

Flesca et al. in [12], by contrast, do not rely on graph matching algorithms. They represent the structure of an XML document as a time series in which each occurrence of a tag corresponds to a given impulse. Thus, they also take into account the order in which tags appear in the documents. Then, by analyzing the frequencies of the corresponding Fourier transform, they can state the degree of similarity between document structures.

These approaches measure the structural similarity between two XML documents, thus their goal is the same of the tree-to-tree transformation problem, and is substantially different from ours, which measures the structural similarity between a tree (the document) and a set of trees, intensionally represented as a DTD. Thus, as discussed above, these approaches could be adopted to measure the structural similarity between a document and a DTD through the extensional approach described in Section 3.

6.2. Structural Similarity among Schemas

The issue of measuring the structural similarity between two schemas has been extensively investigated in the context of heterogeneous data integration [2,11,23,28] and, recently, in the context of clustering of XML DTDs [19].

The problem of heterogeneous data integration is that of identifying corresponding components in different schemas, keeping into account both the names and the structure of schema elements. Thus, the need arises also in that context of analyzing structure similarity.

A recent survey on automatic schema matching proposed a taxonomy of solutions differentiating between schema-and-instance level, element-and-structure level, and language-and-constraint-based matching approaches [8,24]. Furthermore, the distinction between hybrid and composite combination approaches is introduced. An hybrid approach consists in the integration of many matching approaches in the taxonomy in an integrated system, whereas a composite approach exploits different matching approaches and combines their results in a single evaluation. Cupid [20], LSD [10], and COMA [9] are prototypes of data integration systems supporting XML schema matching.

Cupid [20] is an hybrid approach that considers both tag names and hierarchical structures of schema. The similarity between an element of the first schema and an element of the second schema relies on the similarity of their components hereby

emphasizing the name and data type similarities present at the finest granularity level (leaf level). LSD (Learning Source Description) [10] is a composite approach based on machine learning techniques. The application of such a system requires a *training phase* which can incur a substantial manual effort. Finally, COMA [9] is a composite approach, which provides an extensible library for the application of different approaches and supports various ways for combining matching results.

Despite their differences these approaches map a schema (expressed through a DTD or an XML schema) into an internal schema. This internal schema is more similar to a data guide for semi-structured data [14] than to a DTD. Therefore, constraints on the occurrences of an element or group of elements are not considered in performing the structural match. In our matching algorithm, by contrast, we consider both optional and repeatable elements as well as alternative of elements. Moreover, our matching algorithm differs from them because it consider the match between a value (i.e. the document) and a type (i.e. a DTD) and the presence of the `ANY` and `EMPTY` types.

In [19] the authors propose *XClust*, an integration strategy that involves the clustering of DTDs. A matching algorithm based on the semantics, immediate descendent and leaf-context similarity of DTD elements is developed. The matching algorithm analyzes element by element in order to identify possible matches among direct subelements, considering the cardinality of the elements (optional, repeatable or mandatory) and the similarity of their tags. The internal representation of DTDs is more sophisticated than the integration systems presented above. However, they do not consider DTDs that specify alternative elements as our matching algorithm does.

6.3. Structural Similarity among Data and Schema

To the best of our knowledge, almost no approach has been developed to measure structure similarity, despite the practical relevance of the problem, both from a data modeling and a querying perspective. The only approaches we are aware of are by Grahne and Thomo in [15] and by us in [5].

In [15] the problem of determining whether semi-structured data, represented as edge-labeled graphs, approximately conform to a data guide is mentioned. The focus of this approach is, however, on approximate querying: starting from a regular path query and a regular *transducer*[7] specifying the allowed sequences of elementary "distortions", the answers that are within an approximation of the original query are determined. The transducer also defines a function for the distance from the original query. That paper also discusses how the approach can be used for detecting whether data instances approximately satisfy a schema such as a data guide. A basic assumption in such work is that users specify a distortion transducer, through which they can distort the data guide through allowed elementary distortions and then test if the database conforms to the distorted data guide. A notion of k-satisfaction is also introduced meaning that k is the bound of the distance between the database and the data guide, thus providing a quantitative measure of approximate satisfaction. The main difference between that approach and the one we propose in this paper is that the former is based on the assumption that the possible deviations from the original schema specification and their importance (weight) are specified by users through the distortion transducer, whereas we assume no a-priori knowledge of possible deviations. Moreover, the approach in [15] has not been developed for XML documents and DTDs, but for generic semi-structured data and data guides represented as edge-labeled graph. A data guide, however, is simpler than a DTD since it does not contain constraints on the repeatability or alternativeness of elements. Finally, no experimental results assessing the practical effectiveness and efficiently of the approach have been reported.

In [5] we proposed an approach to automatic object classification. This approach is based on an object-oriented data model whose type sys-

[7] A transducer (S, I, O, τ, s, F) is a finite set of states S, an input alphabet I, an output alphabet O, a starting state s, a set of final states F, and a transition-output function τ from finite subsets of $S \times I^*$ to finite subsets of $S \times O^*$.

tem has been extended to handle semi-structured data, for instance with union types. In this approach a predefined database schema exists but objects are allowed to be created without associating them with a class of the schema. For these objects, the class that most closely describes the object structure is automatically determined. The notion of membership is weakened in *weak membership*, only requiring that the components in the object state be a subset of the components of the class. Since an object can be weak member of several classes, two measures are employed to determine the *most appropriate* class for an object, among the ones of which the object is a weak member. The *conformity degree* measures the similarity degree between the type of the semi-structured object and the type of the class, and the *heterogeneity degree* of the class measures how much the extension of the class is heterogeneous. Besides the differences resulting from the different underlying models (an ad-hoc object data model for semi-structured data rather than XML[8]), the main differences between that approach and the one described in this paper are that in [5] only *minus* elements are considered (weak membership does not allow *plus* elements) and that tag similarity is not considered (weak membership requires that attribute names be identical).

7. Concluding Remarks

In this paper we have proposed a matching algorithm for measuring the structural similarity between an XML document and a DTD. Moreover, some applications of the matching algorithm have been presented for the classification of XML documents against a set of DTDs, the evolution of DTD structures, the evaluation of structural queries, the selective dissemination of XML documents, and the protection of document contents.

We are currently investigating some extensions of the matching algorithm along different directions. For what concerns the matching algorithm, we are extending it for taking into account the new features of XML schema with respect to DTDs (definition of subtypes, constraints on the repetition of elements, a richer set of types, etc.). Moreover, for what concerns applications we are developing a tool which integrates all the features illustrated in the paper. Finally, we are currently exploiting the similarity measure as an alternative approach for clustering XML documents.

[8] A relevant difference is that collection values are explicitly described in the object type system, whereas in XML no explicit collection value is given and thus they need to be discovered.

REFERENCES

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
2. C. Batini, M. Lenzerini, and S. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
3. E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Specifying and Enforcing Access Control Policies for XML Document Sources. *World Wide Web Journal*, 3(3), 2000.
4. E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Protection and Administration of XML Data Sources. *Data and Knowledge Engineering*, 43(3):237–260, 2002.
5. E. Bertino, G. Guerrini, I. Merlo, and M. Mesiti. An Approach to Classify Semi-Structured Objects. In *Proc. of 13th European Conf. on Object-Oriented Programming (ECOOP)*, LNCS(1628), pages 416–440, 1999.
6. E. Bertino, G. Guerrini, and M. Mesiti. Measuring the Structural Similarity among XML documents. Technical Report DISI-TR-02-02, University of Genova, 2001. Submitted for publication.
7. E. Bertino, G. Guerrini, M. Mesiti, and L. Tosetto. Evolving a Set of DTDs According to a Dynamic Set of XML Documents. In *XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshop Revised Papers*, LNCS(2490), pages 45–66, 2002.
8. H.-H. Do, S. Melnik, and E. Rahm. Comparison of Schema Matching Evaluations. In *Web, Web-Services, and Database Systems*, LNCS(2593), pages 221–237, 2003.

9. H.-H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *Proc. of 28th Int'l Conf. on Very Large Databases (VLDB)*, pages 610–621, 2002.
10. A. Doan, P. Domingos, and A. Y. Halevy. Reconciling Schemas of Disparate Data Sources: a Machine-Learning Approach. *SIGMOD Record*, 30(2):509–520, 2001.
11. A. K. Elmagarmid and C. Pu. Guest Editors' Introduction to the Special Issue on Heterogeneous Databases. *ACM Computing Surveys*, 22(3):175–178, 1990.
12. S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Detecting Structural Similarities between XML Documents. In *Proc. of 5th Int'l Workshop on the Web and Databases*, 2002.
13. M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In *Proc. of Int'l Conf. on Management of Data (SIGMOD)*, pages 165–176, 2000.
14. R. Goldman and J. Widom. Dataguides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. of 23rd Int'l Conf. on Very Large Data Bases (VLDB)*, pages 436–445, 1997.
15. G. Grahne and A. Thomo. Approximate Reasoning in Semi-structured Databases. In *Proc. of 8th Int'l Workshop on Knowledge Representation meets Databases*, volume 45 of *CEUR Workshop Proceedings*, 2001.
16. J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting Semistructured Information from the Web. In *Proc. of Workshop on Management of Semistructured Data*, Tucson, 1997.
17. J. Hipp, U. Guntzer, and G. Nakhaeizadeh. Algorithms for Association Rule Mining- a General Survey and Comparison. *SIGKDD Explorations*, 2(1):58–64, 2000.
18. G. Lee, K. Lee, and A. Chen. Efficient Graph-Based Algoritms for Discovering and Maintaining Association Rules in Large Databases. *Knowledge and Information System*, 3(3):338–355, 2001.
19. M. Lee, L. Yang, W. Hsu, and X. Yang. XClust: Clustering XML Schemas for Effective Integration. In *Proc. of 11th Int'l Conf. on Information and Knowledge Management (CIKM)*, pages 292–299, 2002.
20. J. Madhavan, P. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proc. of 27th Int'l Conf. on Very Large Databases (VLDB)*, pages 49–58, 2001.
21. M. Mesiti. *A Structural Similarity Measure for XML Documents: Theory and Applications*. PhD thesis, University of Genova, Italy, 2002.
22. A. Nierman and H. Jagadish. Evaluating Structural Similarity in XML Documents. In *Proc. of 5th Int'l Workshop on the Web and Databases*, 2002.
23. C. Parent and S. Spaccapietra. Issues and Approaches of Database Integration. *Communications of the ACM*, 41(5):166–178, 1998.
24. E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.
25. S. V. Rice, H. Bunke, and T. A. Nartker. Classes of Cost Functions for String Edit Distance. *Algorithmica*, 18(2):271–280, 1997.
26. G. Salton, C. Yang, and C. Yu. A Theory of Term Importance in Automic Text Analysis. *Journal of the American Society for Information Sciences*, 26(1):33–44, 1975.
27. S. M. Selkow. The Tree-to-Tree Editing Problem. *Information Processing Letters*, 6(6):184–186, 1977.
28. A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
29. I. Stanoi, G. Mihaila, and S. Padmanabhan. A Framework for the Selective Dissemination of XML Documents based on Inferred User Profiles. In *Proc. of 19th Int'l Conf. on Data Engineering*, 2003.
30. A. Tversky. Features of Similarity. *Journal of Psychological Review*, 84(4):327–352, 1977.
31. W3C. Document Object Model (DOM), 1998.

32. W3C. Extensible Markup Language (XML), 1998.
33. W3C. XML Path Language (Xpath), 1999.
34. K. Zhang and D. Shasha. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM Journal of Computing*, 18(6):1245–1262, 1989.
35. K. Zhang, R. Statman, and D. Shasha. On the Editing Distance Between Unordered Labeled Trees. *Information Processing Letters*, 42(3):133–139, 1992.